CSS Grid documentation

****

********** Overview of the CDAT interface to cssgrid **********

****

INTRODUCTION TO NGMATH

The ngmath library is a collection of interpolators and approximators for one−dimensional, two−dimensional
and three−dimensional data. The packages, which were obtained from NCAR, are:

natgrid −− a two−dimensional random data interpolation package based on Dave Watson's nngridr.

dsgrid −− a three−dimensional random data interpolator based on a simple inverse distance weighting algorithm.

fitgrid −− an interpolation package for one−dimensional and two−dimensional gridded data based on Alan Cline's Fitpack. Fitpack uses splines under tension to interpolate in one and two dimensions.

csagrid −− an approximation package for one−dimensional, two−dimensional and three−dimensional random data based on David Fulker's Splpack. csagrid uses cubic splines to calculate its approximation function.

cssgrid −− an interpolation package for random data on the surface of a sphere based on the work of Robert Renka. cssgrid uses cubic splines to calculate its interpolation function.

shgrid −− an interpolation package for random data in 3−space based on the work of Robert Renka. shgrid uses a modified Shepard's algorithm to calculate its interpolation function.

COMPARISION OF NGMATH PACKAGES

Three−dimensional packages −− shgrid, csagrid and dsgrid.

shgrid is probably the package of choice for interpolation. It uses a least squares fit of biquadratics to construct its interpolation function. The interpolation function will pass through the original data points.

csagrid uses a least squares fit of cubic splines to calculate its approximation function: the calculated surface will not necesarily pass through the original data points. The algorithm can become unstable in data
sparse regions.

dsgrid uses a weighted average algorithm and is stable in all cases, but the resultant interpolation is not usually smooth and execution time is very slow. dsgrid is probably best used when csagrid and shgrid

fail or for comparative purposes.

Two−dimensional packages −− natgrid, fitgrid, csagrid and dsgrid.

natgrid is the package of choice in most cases. It implements a very stable algorithm and has parameters
for adjusting the smoothness of the output surface.

fitgrid offers user−settable parameters for specifiying derivatives along the boundary of the output grid which are not available in natgrid.

csagrid produces an approximate two−dimensional surface which may be smoother than that produced by fitgrid
and natgrid.

dsgrid is not recommended for two−dimensional surfaces. natgrid is superior in all respects.

One−dimensional packages −− fitgrid and csagrid.

fitgrid is definitely the package of choice. It has many features not available in csagrid, such as interpolating parametric curves, finding integrals, handling periodic functions, allowing smoothing that
varies from linear to a full cubic spline interpolation and specifying slopes at the end points.

Interpolation on a sphere −− cssgrid.

cssgrid is designed specifically for interpolating on a sphere. It uses cubic splines to calculate an interpolation function.

CSSGRID PACKAGE

cssgrid is the only algorithm in the ngmath library specifically designed for interpolation on a sphere. cssgrid implements a tension spline interpolation algorithm to fit a function to input data. Tension splines are an extension of cubic splines whereby a user−specified tension factor controls the fitting function from essentially linear interpolation to pure cubic spline interpolation. The input data is specified on a randomly spaced set of latitude−longitude coordinates.

There are entries in the cssgrid package for: interpolating random data on a sphere to a user−specfied grid,
converting between latitude−longitude coordinates and the equivalent Cartesian coordinates on a unit sphere,
for calculating Delaunay triangulations, for calculating Voronoi polygons, and for setting and retrieving user−specified control parameters. Values of the default control parameters should suffice in some cases.

CSSGRID CONTENTS

Access through Python to the cssgrid package from NCAR's ngmath distribution is provided directly through the module
cssgridmodule.so which was generated by connecting the Fortran routines to Python using the Pyfort implementation
by Paul Dubois.

REQUIRED FILE

cssgridmodule.so −− the Python interface to the ngmath cssgrid package.

USEFUL FILES

css.py −− the object oriented interface including a general help package.
cssgridtest.py −− the code to test css.py and to write documentation.

USAGE

This module is designed to use in two ways. One is through the use of the object oriented interface to the underlying
functions. This approach is recommended for users not already familiar with the original cssgrid distribtution because
it simplifies the calls to the routines. The other method uses the original functions calling them directly from Python.

−−−−−−−−−−−−−−−−−− OBJECT ORIENTED APPROACH −−−−−−−−−−−−−−−−

The css module contains the Cssgrid class and its single method, rgrd, which provides access to all the cssgrid
functions (except css2c and csc2s which convert from spherical to cartesian coordinates available as utilities
directly from the cssgridmodule as noted below). The object oriented approach has been organized as a two step
process.

STEP 1.

To make an instance, r, type:

```
import css

r = css.Cssgrid(lati, loni, lato, lono)

or

r = css.Cssgrid(lati, loni)
```

where lati, loni and lato, lono are the input and output grid coordinate arrays in degrees.

The input grid must be organized in a list format. The size of the lati array and the loni array are necessarily equal. For example, if there are n input data points randomly spaced on the sphere, there are n latitudes and n longitudes. The output grid coordinate arrays describe a rectangular grid on the sphere.

The choice between the two examples is made according to requirements in subseqent calls to the method
function. The first choice is necessary for the interpolation from the input to the output grid. The

latter
choice is sufficient if the subsequent request is for Delauanay triangles or Voronoi polygons which require
only an input grid.

The grid coordinate arrays can be single precision (Numeric.Float32) or double precision (Numeric.Float64). The
decision on whether to call for a single or a double precision computation subsequently is made by looking at
the type of these arrays.

To look at the default settings for the control parameters and a brief description of thier properties, type

r.printDefaultParameterTable()

To change a setting type the new value. For example, to set sig to 0.5, type

r.sig = 0.5

To find a value without printing the table, type the name. For example, to exam the value of nsg, type

r.nsg

To check the settings type

r.printInstanceParameterTable() −− prints in tabular form the parameters used in subsequent calls to the method
function rgrd.
or

printStoredParameters() −− prints the parameters in memory which may differ from the above if the user
has made more than one instance of the Cssgrid class.

STEP 2.

cssgrid is restricted to two dimensions on the sphere. Consequently, it is the user's responsibility to reduce the
processing of higher dimensional data to a sequence of calls using only two dimensional data.

There are three basic computations provided by the single method function: regridding data on a sphere using Delaunay
triangulations and Voronoi polygons, calculating Delaunay triangulations only and calculating Voronoi polygons only.
Consequently, the user has the following three choices:

1. Compute the interpolation

To interpolate the random data, dataIn, associated with the list arrays (loni, lati) to the rectilinear grid
(lono, lato), type

dataOut = r.rgrd(dataIn)

where dataOut is the interpolated data on the (lono, lato) grid.

2. Compute the Delaunay triangulations

To find the triangulation associated with the grid (loni, lati), type

ntri = r.rgrd(compType = 'triangles') where

ntri −− a 2D integer array containing the nodes of the triangles in the triangulation. The nodes for the jth triangle are in the triple composed of ntri(j,1), ntri(j,2) and ntri(j,3). These indices reference the sequence in the input coordinate grid. For example, if the triple (5,1,2) were in the
list of triples, it would describe the triangle having vertices at (loni(5), lati(5)), (loni(1), lati(1)) and(loni(2),lati(2)).

3. Compute the Voronoi polygons

To find the polygons associated with the grid (loni, lati), type

latV, lonV, rc, nv = r.rgrd(compType = 'polygons', indexVoro = ni, firstCall = nf)

where the meaning of the input arguments are as follows:

ni −− the index of the input coordinate for which you want to determine the Voronoi polygon. The lowest
value is 0.

nf −− a flag indicating if this is the first call to retrieve Voronoi polygons for this dataset(1=yes,0=no).
Calls subsequent to the first call for a given dataset are much faster than the first call.

where the meaning of the output tuple is as follows:

latV −− an array of latitude values in degrees for the Voronoi indices. (These are circumcenters of
circles passing through the Delaunay triangles. If a coordinate is a boundary point, then the circle may pass through certain "pseudo points" that have been added to the original dataset in
order to complete the Voronoi polygon.)

lonV −− an array of longitude values in degrees for the Voronoi indices.

rc −− Array containing the arc length (in degrees) of the angle between a circumcenter and its associated
triangle vertices.

nv −− an array containing indices for the Voronoi polygon enclosing the coordinate (loni(ni), lati(ni)).

The indices returned in this array refer to the coordinates returned in latV and lonV. For example,
if the integer "j" is an element of the nv array, then (lonV(j),latV(j)) is a vertex of the Voronoi
polygon enclosing (loni(ni), lati(ni)). The indices in nv list out the vertices of the Voronoi polygon
in counter−clockwise order.

In all three cases, a call to rgrd automatically sets the values of the control parameters for use in the computation
to those set in the instance either as the defaults or by explicit choice by the user.

−−−−−−−−−−−−−−−−−− ORIGINAL FUNCTION APPROACH −−−−−−−−−−−−−−−−−

The module cssgridmodule.so exports the following functions to Python from the original Fortran library:

Single precision procedures:
- cssgrid −− interpolation on a sphere
- cssgtri −− calculates a Delaunay triangulation
- csvoro −− calculates Voronoi polygons
- csseti −− sets integer parameter values
- csgeti −− retrieves values for integer parameter values
- cssetr −− sets real parameter values
- csgetr −− retrieves values for real parameter values

Double precision procedures:
- cssgridd −− interpolation on a sphere
- cssgtrid −− calculates a Delaunay triangulation
- csvorod −− calculates Voronoi polygons
- cssetd −− sets double precision parameter values
- csgetd −− retrieves values for double precision parameter values

In addition it contains:
- css2c −− converts lat/lon to Cartesian coordinates
- csc2s −− converts Cartesian to lat/lon coordinates

Information on the use of the routines is available by importing cssgridmodule and printing the docstring
of interest. For example, documentation for the routine cssgrid is obtained by typing

```
import cssgridmodule
print cssgridmodule.cssgrid.__doc__
```

This same information is available in the help package.

A description of the control parameters is not in the cssgridmodule documentation. It can be found by typing

```
import css
css.printParameterTable()
```

The documentation associated with the cssgridmodule.so, such as the doctrings, describe the Fortran code. The
indices start with 1 (not 0 as in the Python interface) and the arrays are transposes as nomenclature. For
example, the Fortran array dimensioned as (number of latitudes, number of longitudes) is the same physically
as the Python or C array dimensioned as (number of longitudes, number of latitudes). Since the calls are from
Python the array indices start with 0. It is only in a request for a specific index submitted to Fortran that
that the indices start at 1 .

The long and the short of this is as follows. If you use the interface, indices start at 0. If you call the routines
directly, in specific requests indices start at 1. In either case, data is passed with the latitude index moving
faster than the longitude index.

DOCUMENTATION

Documentation is provided through Python's docstrings, essentially Python style program comments. A help package
provides instructions on the use of cssgrid. A table of contents is printed to the screen by typing

```
css.help()
```

after importing css.

A hard copy of this documentation is written to the file cssgridmodule.doc after import cssgridtest by typing

```
cssgridtest.document()
```

TESTING

To run a test of the interpolation, the Delaunay triangulations and the computation of the Voronoi polygons, in single
precision and double precision, and to get a copy of this documentation, type

```
cdat cssgridtest.py
```

HELP PACKAGE EXAMPLE

********** Default Parameter Table **********

---

name	type	legal values	default	description
sig	float	>= 0.	1.0	value of tension factor for splines (0. is for cubic)
tol	float	> 0.	0.01	tolerance in making gradient differences to terminate iteration for global gradients
ttf	float	> 0.	0.01	tolerance in determining accuracy of tension factor to approximate optimum value
nls	int	>= 4	10	number of nodes to use in the least squares fit
nsg	int	>= 2	10	max number of iterations to use in computing automatic tension factors
isg	int	any	0	0 to revert to calculating automatic tension factors rather than using a constant
igr	int	any	1	flags use of global or local gradients (1 for global, anything else for local)
mvl	float	any	−8.0	used by NCL functions